

REMARKS

Claims 1-30 are pending in this application. Claims 10, 12, and 16 have been amended. Claims 13, 14, 19, 24, and 28 have been cancelled.

The Applicant thanks the Examiner for accepting the drawings and terminal disclaimer previously submitted. The Applicant also thanks the Examiner for conducting a personal interview with the Applicant and his representatives on September 12, 2006. During that interview, the phrase "in real time" from claim 14 was discussed and compared with the "background translator 34" of Yates. The Examiner and Applicant agreed to some extent as to the "specialized partial IR" of claim 13, in particular the differences between the subject matter of this claim and the cited art (Yates and Chambers). In this Response, the Applicant reiterates the arguments and positions set forth during the interview.

Claims Rejections – 35 USC § 102

At paragraph 6, the Examiner rejects claims 13, 19, 24 and 28 under 35 USC § 102(e) as being anticipated by U.S. Pat. No. 6,427,234 (Chambers). At paragraph 7, the Examiner rejects claim 14 under 35 USC § 102(e) as being anticipated by U.S. Pat. No. 5,930,509 (Yates). The Applicant respectfully disagrees with these rejections. To expedite prosecution, however, the Applicant cancels claims 13, 14, 19, 24, and 28 without prejudice, reserving the right to pursue those claims in a subsequent application. Accordingly, this rejection should be withdrawn.

Claims Rejections – 35 USC § 103

At paragraph 9, the Examiner rejects claims 1-12, 15-18, 20-23, 25-27, 29 & 30 under 35 USC § 103(a) as being unpatentable over Yates in view of Chambers. The Applicant respectfully disagrees with the analysis of Yates and Chambers as presented particularly at paragraph 9 of the Office Action, and respectfully submits that the claimed invention is not obvious in view of the cited prior art.

Yates

Yates uses intermediate representation as part of a background translation process 34 and thereafter retrieves the translated code from a cache managed by a server process 36 whenever the same portion of code is encountered subsequently.

In more detail, Yates discloses a binary image conversion system where a run-time system 32 converts non-native binary code into native executable code through an interpreter (see column 8, line 20 “the run-time system initiates an interpretation process”) and the run-time system 32 gathers “profile data or statistics” in response to execution of the native image which are fed to a server process 36 (see column 8, lines 1-12). Thereafter, a background system 34 receives the profile data generated by the run-time system 32 and, in accordance with the characteristics of the profile data, translates certain portions of the non-native image 17b into a native binary image 17c.

Thus, in Yates, a particular portion of program code is executed through the interpreter of the run-time system 32 and then is translated by the background process 34. Like many prior art translator systems, the translated code produced by the background process 34 is stored in a code cache held by the server process 36. Whenever that same portion of program code is encountered for a second or subsequent time then the translated native image is retrieved from the cache by the server process 36. This view of Yates is reinforced by column 9, lines 19-34 and 39-52 cited in the Office Action. In particular, column 9 makes absolutely clear that once a portion of code has been translated through the background system 34, it is not translated again subsequently:

“...Each time there is a new execution of the application program stored in segment 17b, the run-time system will send a request to the server process 36 for native code corresponding to the non-native code currently in the run-time system 32. ... **The server process 36 supplies corresponding translated code (if any) to the run-time system 32. If there is translated code, the run-time system 32 will have the translated code execute in place of interpreting the code. ... This process will be ongoing until all portions of the non-native image have been encountered by the user and all of the portions which can be translated by the background system 34 have been translated.**” [Yates, column 9, lines 19-30 and 49-52, emphasis added]

The run-time system 32 does not use intermediate representation (see especially the detailed description of the Run-Time Interpreter at column 13 line 44 to column 18 line 31). Instead, the only part of the conversion system in Yates that describes using intermediate representation is in relation to the background translator process 34 (see especially column 57 lines 1 to 12)

Yates clearly specifies that a single IR is to be used throughout the translation and optimization process performed by the background system 34. As noted at column 2, lines 13-18 and column 70, lines 7-21 and column 73, lines 26-41 as cited in the Office Action, and

elsewhere within Yates, the translation performed by the background process 34 uses intermediate representation. However, we can also see that the specific manner in which the intermediate representation is generated in Yates is an important aspect of that disclosure. Thus, there is a strong teaching for the skilled person not to make any changes to the IR used by the system of Yates.

Finally, the detailed discussion of the translation process performed by Yates at column 46, line 20 to column 47, line 25, specifically contrasts the binary translation and optimization of Yates against the very different environment of a compiler:

“Difficulties arise when performing procedural and interprocedural optimizations on a binary image, because traditional assumptions cannot be made about its structure.... As a result, a new set of problems evolves when implementing procedural and interprocedural optimizations in the background optimizer 58 that optimizes a binary image since assumptions about its structural cannot be made. **Existing procedural and interprocedural optimization techniques typically implemented in an optimizing compiler cannot readily be employed in the background optimizer 58 because properties and program structure about the code included in the binary image input cannot be assumed.**” [Yates, column 46, lines 56-58; column 47, lines 3-11, emphasis added]

Chambers

Chambers discloses a selective dynamic compiler. The compiler of Chambers compiles from a high-level human oriented programming language into machine-executable instructions. As explained in columns 1-3 of Chambers, most portions of the program are compiled statically in advance, whilst some selected portions are compiled dynamically at run-time. Here, the selective dynamic compiler of Chambers relies on the high-level program being annotated by a programmer in advance in order to determine those portions of program that are to be compiled dynamically. See column 2, lines 13-32, column 4, lines 26-28 and column 5, lines 1-4.

Chambers specifically states that a human programmer must manually perform the annotations, and it is not possible for the annotations to be performed automatically. See column 2, lines 7-11. Thus, it is impossible for the compiler of Chambers to receive as input anything other than a high-level program with annotations added by a human programmer.

Secondly, referring to column 8, lines 15-20, Chambers specifically states that the selective dynamic compiler does not use intermediate representation of any form:

... a custom dynamic compiler for each dynamic region (also called a generating extension) is built simply by inserting emit code sequences into the set-up code for each instruction in the template code; the template sub-graph is then discarded. **This dynamic compiler is fast, in large part, because it neither consults an intermediate representation nor performs any general analysis when run.** Instead, these functions are in effect "hard-wired" into the custom compiler for that region, represented by the set-up code and its embedded emit code. [Chambers, column 8, lines 15-24, emphasis added]

As noted in the Office Action, Chambers discloses dynamically compiling the program code based on prevailing run-time conditions, referring particularly to column 9, lines 15-27 & 33-43. That is, Chambers discloses creating specialized program code, but only in the context of a selective dynamic compiler.

Yates in view of Chambers

As discussed above, Yates uses intermediate representation as part of a background translation process 34 and thereafter retrieves the translated code from a cache (server process 36) whenever the same portion of code is encountered subsequently.

Hence, it appears that Yates only discloses the following elements of claim 1:

A method of generating an intermediate representation of program code, the method comprising the computer implemented steps of:

[during] translation of a given portion of program code, generating intermediate representation which is required to execute that portion of program code [for all possible conditions];

We respectfully submit that Yates does not disclose the following underlined portions of claim 1:

A method of generating an intermediate representation of program code, the method comprising the computer implemented steps of:

on an initial translation of a given portion of program code, generating and storing only intermediate representation which is required to execute that portion of program code with a prevailing set of conditions; and

whenever subsequently the same portion of the program code is entered, determining whether the prevailing set of conditions are now different, subsequent conditions; and

if no intermediate representation compatible with the subsequent conditions has previously been generated, generating additional intermediate representation required to execute said portion of program code with said subsequent conditions.

Chambers concerns a selective dynamic compiler that compiles from a programmer-annotated high-level programming language directly into machine-executable instructions without using any form of intermediate representation, and compiles specialized program code according to prevailing run-time conditions. Thus, Chambers teaches a method of creating specialized program code only in the context of a selective dynamic compiler.

We respectfully submit that (a) Chamber does not fulfill the deficiencies in Yates as identified above, and (b) one skilled in the art cannot combine Yates with Chambers in the manner suggested in the Office Action.

It is not possible to combine the binary-to-binary conversion system of Yates with the high-level to binary compiler system of Chambers because, as discussed above, (i) the inputs to these two systems are mutually incompatible and (ii) the operation environments of these two systems are also mutually incompatible.

Further, both Chambers and Yates specifically teach away from making any such combination. In fact, Chambers specifically mentions at column 1, lines 63-67 that dynamic compilation is itself possibly a solution to the problem of portability across different hardware architectures which would, of course, obviate the need for binary image conversion as in Yates. Thus Chambers specifically highlights that compilation is a fundamentally different strategy compared to binary image conversion as in Yates and teaches away from any possible combination with Yates. Conversely, Yates specifically teaches at column 46, line 20 to column 47, line 25 that the translation and optimization process performed by the background process 34 is very different to the environment of a compiler.

Finally, the combination of Chambers with Yates still does not recite all of the limitations set out in claim 1 and noted above. In particular, there is still no teaching or suggestion to a person of ordinary skill in the art at the time the invention was made to supplement Yates with the program code specialization features of Chambers, such that Yates generates and stores intermediate representation for only the condition prevailing at the time that a portion of program code is entered, and generates additional intermediate representations as needed for different, subsequent conditions. That is, the code specialization of Chambers still

does not lead the skilled person to make a specialization of the intermediate representation as in the claimed invention.

Independent claim 11

With respect to claim 11, Yates does not disclose generating in the intermediate representation a plurality of “register objects” or “expression objects” as recited in the claim. Further, Yates does not disclose that the intermediate representation is generated and stored for a block program code. Further, Yates does not disclose that the intermediate representation is subsequently re-issued if the same block of program code is later re-entered. Clearly, as discussed above for claim 1, the intermediate representation of Yates is used only once during the sole translation process by the background translator 34. There is no need in Yates for the intermediate representation to be stored, and there is no need in Yates for the intermediate representation to be subsequently re-issued.

As correctly identified in the Office Action, Yates does not disclose that at least one block of program code can have alternative unused entry conditions or effects or functions and said intermediate representation is only generated and stored as required to execute that block of program code with a then prevailing set of conditions (i.e. specialized intermediate representation). Further, as noted above, the selective dynamic compiler of Chambers cannot be combined with the binary conversion system of Yates and, even if such a combination could be made, Chambers explicitly does not use intermediate representation and instead teaches specialization of the compiled program code. There is no teaching in either document which would lead the skilled person to arrive at special-case intermediate representation as set out in claim 11. This is a fundamental difference, and represents a seismic shift in the performance of a binary image conversion system using the method of the claimed invention.

One skilled in the art whom has read and understood the teachings of Yates and Chambers still has no clear teaching or suggestion towards the use of specialized intermediate representation as in the claimed invention. One must of course be careful not to use hindsight when assessing obviousness.

Independent claims 15 & 16

With respect to claim 15 & 16, the systems recited in these claims are allowable for reasons analogous to those given above for claims 1 and 11.

Dependent claims 2-10

Each of the dependent claims is allowable because they depend from an allowable main claim. However, for the Examiner's convenience, we provide brief comments concerning the dependent claims.

With respect to claim 2, neither Yates nor Chambers discloses at least the recited step (b).

With respect to claim 3, Yates only discloses using intermediate representation as part of a background translation process. This is not equivalent to generating the intermediate representation dynamically as the program code is running. Further, neither Chambers nor Yates discloses that at a first iteration of a particular subject code instruction (i.e. input subject code as opposed to translated output target code) where the instruction has a plurality of possible effects or functions, generating and storing special case intermediate representation representing only the specific functionality of that particular subject code instruction required at that iteration or, for each subsequent iteration of the same subject code instruction, determining whether special case intermediate representation has been generated and if not then generating additional special case intermediate representation specific to the newly determined functionality of the subject code instruction as claimed.

With respect to claim 4, neither Chambers nor Yates discloses associating a test procedure with a respective subject code instruction.

With respect to claim 5, Chambers at column 30, lines 37-54 does not discuss any form of subordinate relation for associated test procedures.

With respect to claim 6, Yates at column 71, lines 13-25 concerns the selection of "translation units" which is a unit of code analogous to a routine (see column 47, lines 29-50).

There is no disclosure in Yates or Chambers of optimising the intermediate representation by adjusting the ordering of the test procedures as recited in claim 6.

With respect to claims 7 and 8, the translation of Yates is not performed dynamically as the program code is run. The translation of Yates is clearly performed as a background process 34 separate to and subsequent to the run-time system 32. The background process 34 always runs after the run-time system 32 has interpreted a particular section of code. Even if the background process 34 makes use of idle periods while the run-time system 32 is not working, the background system 34 must always wait until the profile characteristics have been collected from the runtime system 32 before beginning the background translation process. Here, reference was made at interview to column 8, lines 42 to 57 which confirms this understanding of Yates.

With respect to claims 9 and 10, Yates concerns a binary image conversion system whilst Chambers concerns a selective dynamic compiler. Neither document discloses optimising program code written for execution by a processor of a first type so that the program code may be executed more efficiently by the processor of the first type.

Dependent claim 12

With respect to claim 12, neither Chambers nor Yates discloses determining whether a previously stored intermediate representation was for a now currently prevailing set of conditions and if not then generating and storing additional intermediate representation as required to execute a block of program code for the new now currently prevailing set of conditions as recited in this claim.

Dependent claims 17, 18, 20-23, 25, 26, 27, 29 & 30

With respect to claims 17, 18, 20-23, 25 & 26, Yates does not disclose that the intermediate representation is generated at run-time. The run-time system 32 of Yates does not use intermediate representation. Intermediate representation is only used in the translation performed by the background process 34.


With respect to claims 27, 29 and 30, for the same reasons as discussed above Yates and Chambers do not disclose the specialized nature of the intermediate representation as recited in these dependent claims.

Filed herewith is a Request for a Three-Month Extension of Time, which extends the statutory period for response to expire on December 19, 2006. Accordingly, Applicant respectfully submits that this response is being timely filed.

In view of the above amendment, applicant believes the pending application is in condition for allowance. No other fees are believed to be due in connection with the filing of this response, however the Commissioner is authorized to debit Deposit Account No. 08-0219 for any required fee necessary to maintain the pendency of this application.

Respectfully submitted,

Dated: December 14, 2006

A handwritten signature in black ink, appearing to read 'R. Demsher', is written over a horizontal line.

Ronald R. Demsher
Registration No.: 42,478
Attorney for Applicant(s)

Wilmer Cutler Pickering Hale and Dorr LLP
60 State Street
Boston, Massachusetts 02109
(617) 526-6000 (telephone)
(617) 526-5000 (facsimile)